(19) KOREAN INTELLECTUAL PROPERTY OFFICE

KOREAN PATENT ABSTRACTS

(11)Publication number: 1020010053875 A
(43)Date of publication of application: 02.07.2001

(21)Application number: 1019990054422
(22)Date of filing: 02.12.1999

(71)Applicant: KOREA ELECTRONICS & TELECOMMUNICATIONS RESEARCH INSTITUTE

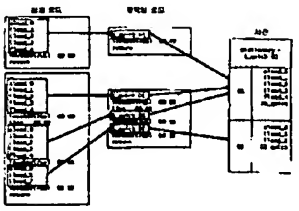
(72)Inventor: BAE, YU SEOK
HAN, TAK DON
KANG, DU JIN
LEE, YEONG MIN
MAENG, HYE SEON
SON, JONG MUN

(51)Int. Cl G06F 9/30

(54) METHOD FOR COMPRESSING BYTE-CODE FOR EMBEDDED JVM

(57) Abstract:

PURPOSE: The method for compressing the byte-code for the embedded JVM(Java Virtual Machine) is provided to reduce the whole size of the byte-code by selecting the code block of a repeated java byte-code, building in a dictionary, and representing the compressed code in the internal of the byte-code.



CONSTITUTION: The method comprises like the following. The repeated code blocks are constituted as the entry of the dictionary. The actual code blocks are deleted in the internal of the byte-code. The repeated code block is stored to the dictionary which composes of the length-fixed entries. The entry of the dictionary consists of a command, which means the compressed code, and of the index information about the dictionary entry. So, the size of the byte-code gets reduced. The 26_quick on the bottom of each code shows the end of compressing the code block.

Legal Status
Date of request for an examination (19991202)
Final disposal of an application (registration)
Date of final disposal of an application (20011130)
Patent registration number (1003197550000)
Date of registration (20011221)

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl. [7]
G06F 9/30

(11) 공개번호 특2001-0053875
(43) 공개일자 2001년07월02일

(21) 출원번호 10-1999-0054422
(22) 출원일자 1999년12월02일

(71) 출원인 한국전자통신연구원
오길록
대전 유성구 가정동 161번지

(72) 발명자 배유석
대전광역시유성구어은동1번지
손종문
대전광역시유성구신성동럭키하나아파트103-802
한탁돈
서울특별시서대문구신촌동134번지연세대학교컴퓨터과학과
맹혜선
경기도성남시분당구정자동정든마을동아아파트104-601
강두진
서울특별시용산구원효로3가238번지
이영민
경기도부천시소사구심곡본1동574-116/2

(74) 대리인 특허법인 신성 박해천
특허법인 신성 원석희
특허법인 신성 최종식
특허법인 신성 박정후
특허법인 신성 정지원

심사청구 : 있음

(54) 내장형 자바가상머신을 위한 바이트코드 압축 방법

요약

본 발명은 내장형 자바가상머신을 위한 바이트코드 압축 방법과 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 관한 것으로, 내장형 자바가상머신에서 자주 반복되는 자바 바이트코드의 코드 블록을 선택하여 사전으로 구성하고 이들에 대한 압축 코드를 바이트코드 내부에서 표현함으로써 바이트코드의 전체 크기를 줄이는 내장형 자바가상머신을 위한 바이트코드 압축 방법과 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터

로 읽을 수 있는 기록매체를 제공하고자 한다. 본 발명은, 클래스파일의 상기 바이트코드에서 반복되는 기본 코드 블록을 파일하여 사전에 저장하는 제 1 단계; 및 상기 기본 코드 블록 위치에 상기 사전에 저장된 상기 기본 코드 블록을 가리키는 압축 코드를 위치시켜 상기 바이트코드를 압축시키는 제 2 단계를 포함하고, 자바 시스템 등에 이용된다.

대표도

도 3

색인어

자바, 바이트 코드, 압축, 내장형 자바가상머신

명세서

도면의 간단한 설명

도 1 은 본 발명이 적용되는 자바가상머신을 탑재한 컴퓨터의 구성예시도.

도 2 는 본 발명에 따른 내장형 자바가상머신을 위한 바이트코드 압축 방법에 있어서 압축코드 표현에 대한 일실시예 설명도.

도 3 은 본 발명에 따른 내장형 자바가상머신을 위한 바이트코드 압축 방법에 대한 일실시예 설명도.

도 4 는 본 발명에 따른 기본 코드 블록의 추출 과정에 대한 일실시예 흐름도.

도 5 는 본 발명에 따른 기본 코드 블록을 사전 엔트리로 만들기 위한 이득값 확인 과정의 일실시예 흐름도.

도 6 은 본 발명에 따른 사전 엔트리 구성 과정에 대한 일실시예 흐름도.

도 7 은 본 발명에 따른 압축 코드 디코딩 과정에 대한 일실시예 흐름도.

*도면의 주요 부분에 대한 부호의 설명

101 : 중앙처리장치 102 : 입출력버스

103 : 출력장치 104 : 램

105 : 롬

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 내장형 자바가상머신을 위한 바이트코드 압축 방법과 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 관한 것으로, 특히 내장형 자바가상머신에서 필요한 메모리 크기를 줄이기 위하여 클래스 파일 중에서 메소드의 바이트코드(bytecode) 부분을 압축하는 방법과 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 관한 것이다.

내장형 기기의 제어나 응용 프로그램으로서 자바 프로그램을 사용하기 위해서는 내장형 기기에 내장형 자바 환경을 탑재시켜야 한다. 내장형 자바 환경은 내장형 자바가상머신과 내장형 자바 응용 프로그래밍 인터페이스(API : Application Programming Interface)로 구성된다.

자바가상 머신은 자바 프로그램의 이진코드인 바이트코드를 직접 수행하는 소프트 프로세서의 역할을 담당한다. 내장형 자바 API는 내장형 프로그램을 작성하기 위해서 기본적으로 제공되어야 하는 자바 프로그램과 네이티브 프로그램이다. 내장형 자바가상머신은 기본적으로 자바가상머신 명세를 따라서 구성된다.

내장형 기기는 메모리 크기에 대한 제약을 가진다. 메모리 크기는 가격에 영향을 미치고 전력소모 및 내장형 시스템의 크기에도 영향을 미치기 때문이다. 따라서, 내장형 시스템은 작은 크기의 제어 및 응용 프로그램을 요구한다. 내장형 시스템을 적은 메모리 상에서도 운용할 수 있도록 하기 위한 여러 방법들이 연구되고 있다.

내장형 자바가상머신을 구현할 때에는 기본적으로 자바가상머신 명세를 따르지만 내장형 기기의 제약점을 고려할 필요가 있다. 내장형 자바가상머신의 특징은 내장형 시스템의 제약점 위에서 수행되어야 하기 때문이다. 따라서, 많은 메모리를 이용하여 빠른 수행 시간을 얻어낼 수 있는 구조보다는 적은 메모리 상에서 효율적으로 수행할 수 있는 구조가 더욱 요구된다.

내장형 기기는 복잡하고 다양한 응용 프로그램이 필요하지 않기 때문에 내장형 기기를 위해 자바 API를 탑재할 때에는, 수행할 응용 프로그램이 직접적으로 이용할 API만을 선별하여 응용 프로그램과 함께 저장한다. 내장형 기기는 일반적으로 디스크를 가지고 있지 않기 때문에 롬 메모리에 적재된다. 내장형 기기에서는 실행 시간에 네트워크나 디스크를 통해서 프로그램을 로딩하는 동적 로딩은 고려되지 않는다. 따라서, 기기를 사용하기 위해서 필요한 모든 프로그램은 런타임 이전에 정해진다.

내장형 기기의 메모리 크기에 대한 제약점을 극복하기 위해서는 내장형 자바가상머신에서 이용할 클래스 파일에 대한 효율적인 압축 방법이 필요하다.

클래스 파일의 여러 구성 요소 중에서 실제 수행 코드에 해당하는 부분을 바이트코드라고 한다. 바이트코드는 자바가상머신 명세에서 정의한 명령어 집합으로 구성된다.

자바가상머신은 스택 머신 구조를 가진다. 따라서, 바이트코드들이 수행하는 모든 연산은 스택을 통해서 이루어진다. 자바가상머신의 또 다른 특징은 수행 중간 값의 저장을 위한 레지스터가 존재하지 않는다는 것이다. 대신 지역 변수 영역을 두어 이를 레지스터 대용으로 이용한다.

바이트코드 내에서 특정 연산을 수행하는 경우, 연산자들은 일반적으로 지역 변수에 위치한다. 따라서, 지역 변수에 저장된 값을 스택으로 푸쉬(push)한 후 연산을 수행하고 그 결과값을 다시 지역 변수에 저장하는 형태로 연산이 수행된다. 따라서, 같은 연산을 수행하는 경우에는 연산 수행에 대한 명령어 전후의 명령어 집합이 같은 명령어들로 구성되는 것이 일반적이다.

바이트코드 수행 중에 메소드를 호출하는 경우에도 유사한 상황이 발생한다. 메소드 호출 시에 필요한 패러미터들은 일반적으로 지역 변수에 저장되어 있다. 메소드 호출을 위해서는 지역 변수에 저장된 값들을 스택으로 푸쉬하고, 메소드를 호출하는 명령어를 수행시킨다. 또한, 메소드 호출 후에 반환되는 결과값은 스택에서 다시 지역 변수로 팝(pop)하여 사용한다. 따라서, 메소드 호출시마다 발생하는 명령어 집합들은 빈번하게 유사한 형태를 보이게 된다.

즉, 자주 반복되는 바이트코드의 블록으로 인해 필요이상의 메모리를 차지하게 되는 문제점이 있다.

발명이 이루고자 하는 기술적 과제

본 발명은 상기한 바와 같은 문제점을 해결하기 위하여 안출된 것으로, 내장형 자바가상머신에서 자주 반복되는 자바 바이트코드의 코드 블록을 선택하여 사전으로 구성하고 이들에 대한 압축 코드를 바이트코드 내부에서 표현함으로써 바이트코드의 전체 크기를 줄이는 내장형 자바가상머신을 위한 바이트코드 압축 방법과 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체를 제공하는데 그 목적이 있다.

발명의 구성 및 작용

상기 목적을 달성하기 위한 본 발명은, 자바 시스템에 적용되는 내장형 자바가상머신을 위한 바이트코드 압축 방법에 있어서, 클래스파일의 상기 바이트코드에서 반복되는 기본 코드 블록을 확인하여 사전에 저장하는 제 1 단계; 및 상기 기본 코드 블록 위치에 상기 사전에 저장된 상기 기본 코드 블록을 가르키는 압축 코드를 위치시켜 상기 바이트코드를 압축시키는 제 2 단계를 포함하는 것을 특징으로 한다.

또한, 본 발명은, 대용량 프로세서를 구비한 자바 시스템에, 클래스파일의 상기 바이트코드에서 반복되는 기본 코드 블록을 확인하여 사전에 저장하는 제 1 기능; 및 상기 기본 코드 블록 위치에 상기 사전에 저장된 상기 기본 코드 블록을 가르키는 압축 코드를 위치시켜 상기 바이트코드를 압축시키는 제 2 기능을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체를 제공한다.

다른 용도의 자바 환경과는 달리 내장형 자바의 경우 런타임 이전에 필요한 모든 클래스 파일들이 정해지므로 실행시간 이전에 압축방법을 적용함으로써 높은 압축 효과를 얻을 수 있다. 클래스 파일의 실질적인 수행코드인 바이트코드를 압축함으로써 메모리에 적재되는 클래스 파일의 크기를 감소시킬 수 있다.

자바 바이트코드는 스택 머신 기반의 머신 코드로 구성되어 있기 때문에 반복되는 기본 블록을 다수 가진다. 따라서, 반복되는 코드 블록을 압축을 의미하는 특정 코드로 대치함으로써 압축 효과를 얻을 수 있다.

코드를 압축하기 위해서는 압축되어야 하는 대상을 선별하는 작업 및 저장하는 방법과 압축된 코드를 표현하는 방법이 제시되어야 한다. 압축 코드를 표현하는 방법은 기존의 코드의 호환성에 지장을 주지 않아야 하며, 압축을 복원하는 시점에서 오버헤드를 최소화할 수 있어야 한다. 또한, 이런 조건 속에서도 높은 압축률을 제공할 수 있어야 한다.

상술한 목적, 특징들 및 장점은 첨부된 도면과 관련한 다음의 상세한 설명을 통하여 보다 분명해 질 것이다. 이하, 첨부된 도면을 참조하여 본 발명에 따른 바람직한 일실시예를 상세히 설명한다.

도 1 은 본 발명이 적용되는 자바가상머신을 탑재한 컴퓨터의 구성예시도이다.

본 발명이 적용되는 자바가상머신을 탑재한 컴퓨터는 중앙처리장치(101), 입출력버스(102), 출력장치(103), 램(RAM. 104) 및 롬(ROM. 105) 등을 포함하여 이루어진다.

자바가상머신은 상기한 구성을 갖는 컴퓨터에 탑재되어 사용자의 프로그램 생성이나 프로그램 동작 요청에 의해 동작하게 된다.

자바가상머신은 어떤 하드웨어 환경인가에 종속적이지 않기 때문에 상기한 컴퓨터 이외에 다양한 하드웨어 환경에서 사용될 수 있다.

도 2 는 본 발명에 따른 내장형 자바가상머신을 위한 바이트코드 압축 방법에 있어서 압축코드 표현에 대한 일실시예 설명도이다

자바 바이트코드는 한 바이트씩 구분되어서 의미를 가진다. 자바 바이트코드의 명령어는 한 바이트, 8비트에 비트 정보를 변화시켜서 서로 다른 명령어를 의미하도록 한다. 모두 256개의 연산코드(opcode)가 존재할 수 있으나 실제로 의미를 가지는 코드는 0번부터 201번까지이다. 자바가상머신 명세에 따르면 203번부터 228번까지의 코드는 퀵(quick) 명령어로 정의하였다. 퀵(quick) 명령어는 동적 로딩을 수행하였을 때 간접 참조를 직접 참조로 변화한 정보를 가지고

명령어를 수행할 수 있도록 재정의하는데 이용된다. 따라서, 내장형 자바가상머신에서는 동적 로딩을 수행하지 않으므로 인해서 간접 참조를 직접 참조를 바꾸는 작업이 런타임 전에 이루어진다. 따라서, 동적 로딩을 지원하지 않는 내장형 자바가상머신에서는 퀵(quick) 명령어를 사용하지 않는다.

압축된 코드를 표현하는 방법으로서 퀵(quick) 명령어를 이용한다. 퀵(quick) 명령어 영역뿐만 아니라 자바가상머신에서 아직 정의하고 있지 않은 메모리 영역을 모두 압축 코드를 표현하는데 사용하는 것도 가능하다. 그러나, 향후 자바 바이트코드의 명령어 집합이 확장되는 경우에는 명령어 충돌이 발생할 수 있기 때문에 내장형 시스템에서는 이용할 필요가 없는 퀵(quick) 명령어 영역만을 이용하도록 한다. 퀵(Quick) 명령어 개수만으로는 구분할 수 있는 엔트리의 수가 제한되기 때문에 피연산자를 1 바이트 추가적으로 이용하도록 한다.

압축 코드로 이용하는 퀵(quick) 명령어는 모두 26개의 퀵(quick)명령어 중에서 25개를 이용한다. 나머지 한 개의 퀵(quick) 명령어는 압축된 코드 블록이 끝나는 정보로 이용된다. 퀵(quick) 명령어에 한 바이트의 피연산자 정보를 부가시켜서 얻을 수 있는 서로 다른 코드 정보의 개수는 [수학식 1]과 같이 구할 수 있다.

수학식 1
(26(퀵 명령어 개수) - 1)*256 ="6400

자바가상머신에서 명령어를 읽어서 해독하는 시점에서 퀵(quick) 명령어 계열의 명령어가 발견되면 다음 바이트에 존재하는 피연산자를 읽어서 실제 코드가 저장된 영역을 계산한다.

이를 도면을 통해 설명하면 나열된 바이트 코드 명령어(201)에서 퀵(quick) 명령어(203)와 피연산자인 인덱스(204)를 포함하여 압축코드(202)를 표현하게 된다.

퀵(quick) 명령어(203)는 도면에 도시된 바와 같이, 203~227번까지의 25개를 사용하며, 피연산자인 인덱스(204)에는 한 바이트를 모두 사용함으로 256개를 사용한다. 따라서, 전체 명령어 개수는 [수학식 1]을 통해 설명한 바와 같이 6400개가 된다.

도 3 은 본 발명에 따른 내장형 자바가상머신을 위한 바이트코드 압축 방법에 대한 일실시예 설명도이다.

바이트코드에서 보여지는 명령어 나열 중 반복되는 코드 블록들을 사전의 엔트리로 구성한 후, 실제 바이트코드 내에서는 코드 블록을 삭제하고 압축된 코드를 의미하는 명령어와 사전 엔트리에 대한 인덱스 정보로 구성함으로써 바이트코드의 크기를 줄이도록 한다. 반복되는 코드 블록은 도 3 에 도시된 바와 같이 고정된 길이의 엔트리로 구성되는 사전에 저장된다.

반복되는 코드 블록의 범위를 기본 코드 블록(Basic Code Block)으로 한정한다. 기본 블록 중에서 해당 블록을 압축함으로써 얻을 수 있는 이득이 큰 블록을 우선적으로 선택하여 사전을 구성한다. 압축으로 인한 이득은 해당 블록이 반복되는 회수와 함께 사전에 저장되는 방식에 따라서 달라진다. 특정 블록이 압축되어 사전의 엔트리로 구성되면 해당 블록의 경우 그 크기의 관계없이 압축 코드를 표현하는데 필요한 2 바이트만이 사용된다. 즉 특정 코드 블록 Bi의 크기가 n 바이트이며 사전 엔트리 크기가 k 바이트(n< ="K)인" 경우 Bi가 프로그램 내에서 m번 반복된다면 이 블록을 압축시킴으로써 얻을 수 있는 이득 G는 다음의 [수학식 2]와 같이 계산될 수 있다. [수학식 2]에서 Bi의 크기 k를 감하여 주는 이유는 사전의 크기가 전체 크기에 반영되어야만 하기 때문이다.

수학식 2
G ="(n - 2)" * m - k

즉, 이득은 원래의 특정 코드 블록의 크기에서 대치되는 2 바이트의 압축코드를 뺀 값에 특정 코드의 반복 횟수 만큼 곱하여 얻은 이득값에 특정 코드에 대해 사전에서 차지하는 크기를 뺀 값이 압축코드에 의해 얻어지는 순수한 이득이다.

도면에서 사전의 내용을 보면 각 코드의 밑에는 26번 퀵(quick) 코드가 놓여 압축된 코드 블록이 끝남을 표시한다.

도 4 는 본 발명에 따른 기본 코드 블록의 추출 과정에 대한 일실시예 흐름도이다.

기본 코드 블록(Basic Code Block)을 추출하는 과정은 다음과 같다.

우선, 클래스 파일에서 코드 필드만을 읽는다(401).

점프 명령어들의 타켓 주소를 기본 블록에서 제외라는 필드에 세팅한다(402).

읽어온 클래스 파일의 코드 필드를 확인하여 기본 코드 블록을 추출한다(403).

추출된 기본 코드 블록을 해쉬 테이블에 넣는다(404).

읽어온 클래스 파일의 코드 필드 중 확인되지 않은 코드 필드가 남아있는지를 판단한다(404). 판단 결과, 확인되지 않은 코드 필드가 남아있으면, 코드 필드를 확인하여 기본 코드 블록을 추출하는 과정(403)부터 반복 수행한다.

확인되지 않은 코드 필드가 남아있는지를 판단한 결과, 남아있지 않으면 기본 코드 블록을 추출하는 과정을 종료한다.

도 5 는 본 발명에 따른 기본 코드 블록을 사전 엔트리로 만들기 위한 이득값 확인 과정의 일실시예 흐름도이다.

기본 블록이 저장된 해쉬 테이블에서 기본 블록을 읽어온다(501).

읽어온 기본 블록이 사전엔트리에 들어갈 수 있는지를 판단한다(502).

판단 결과, 읽어온 기본 블록이 사전엔트리에 들어갈 수 없으면, 기본 블록을 분해하여(503), 해쉬 테이블에 다시 넣고(504), 해쉬 테이블에서 기본 블록을 읽어오는 과정(501)부터 반복 수행한다.

읽어온 기본 블록이 사전엔트리에 들어갈 수 있는지를 판단한 결과, 사전엔트리에 들어갈 수 있으면 상기한 기본 블록의 이득을 계산하여 해쉬 테이블에 저장한다(505).

기본 블록이 저장된 해쉬 테이블에 이득을 계산하지 않은 기본 블록이 있는지를 검사한다(506).

검사 결과, 이득을 계산하지 않은 기본 블록이 있으면 해쉬 테이블에서 기본 블록을 읽어오는 과정(501)부터 반복 수행한다. 이때는 이득을 계산하지 않은 기본 블록을 해쉬테이블에서 불러오게 된다.

기본 블록이 저장된 해쉬 테이블에 이득을 계산하지 않은 기본 블록이 있는지를 검사한 결과, 이득을 계산하지 않은 기본 블록이 없으면 기본 코드 블록을 사전 엔트리로 만들기 위한 이득값 확인 과정을 종료한다.

도 6 은 본 발명에 따른 사전 엔트리 구성 과정에 대한 일실시예 흐름도이다.

기본 블록이 저장된 해쉬 테이블에서 이득이 가장 큰 기본 블록을 선택한다(601).

선택된 기본 블록을 사전에 넣는다(602).

사전에 엔트리를 넣을 공간이 비어있는지를 확인한다(603). 확인 결과, 비어있으면 해쉬 테이블에서 남아있는 기본 블록 중 이득이 가장 큰 기본 블록을 선택하는 과정(601)부터 반복 수행한다.

사전에 엔트리를 넣을 공간이 비어있는지를 확인한 결과, 비어있지 않으면 사전 엔트리를 구성하는 과정을 종료한다.

이때, 사전에 기본 코드 블록을 넣으면서 클래스 파일의 바이트 코드에는 사전에 위치하는 기본 코드 블록을 가르키는 압축 코드로 표현한다.

도 7 은 본 발명에 따른 압축 코드 디코딩 과정에 대한 일실시예 흐름도이다.

자바 바이트 코드를 실행시키기 위해 자바 바이트 코드를 읽어가며 디코딩한다(701).

바이트 코드를 읽어가는 중에 퀵(quick) 명령어를 만나는지를 확인한다(702).

퀵(quick) 명령어를 만나는지를 확인한 결과, 퀵(quick) 명령어를 만났으면 만난 퀵(quick) 명령어가 퀵 리턴(quick return) 명령어인지를 판단한다(703).

판단 결과, 퀵 리턴(quick return) 명령어가 아니면 현재 처리중인 코드 위치를 나타내는 프로그래머블 카운터(PC)를 저장하고, 사전으로 점프한다(704). 점프한 후에는 바이트 코드를 디코딩하는 과정(701)부터 반복 수행한다.

만난 퀵(quick) 명령어가 퀵 리턴(quick return) 명령어인지를 판단한 결과, 퀵 리턴(quick return) 명령어이면 사전의 해당 엔트리에서의 작업이 끝난 것이므로 저장되어 있는 프로그래머블 카운터(PC)의 값에 해당하는 위치로 점프한다(705). 점프한 후에는 바이트 코드를 디코딩하는 과정(701)부터 반복 수행한다.

바이트 코드를 디코딩하면서 퀵(quick) 명령어를 만나는지를 확인한 결과, 퀵(quick) 명령어를 만나지 않고 디코딩을 끝냈으면 바이트 코드를 실행시킨다(706).

즉, 상기한 실시예에 따른 본 발명은 클래스 파일에서 자바 바이트 코드 필드를 읽어와 기본 블록을 추출하여 해쉬 테이블에 저장하고, 기본 블록 해쉬 테이블의 기본 블록들의 이득을 계산하여 이득이 큰 기본 블록부터 사전에 저장한 후, 클래스 파일에 바이트 코드에 대해 압축 코드로 표현하고, 압축 코드로 된 자바 바이트 코드를 디코딩하면서 압축 명령어인 퀵(quick) 명령어를 만나면 사전에 등록된 바이트 코드로 대치하여 디코딩을 수행하고 바이트 코드가 실행되도록 한다.

이상에서 설명한 본 발명은 전술한 실시예 및 첨부된 도면에 의해 한정되는 것이 아니고, 본 발명의 기술적 사상을 벗어나지 않는 범위 내에서 여러 가지 치환, 변형 및 변경이 가능하다는 것이 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 있어 명백할 것이다.

발명의 효과

상기한 바와 같은 본 발명은, 내장형 자바가상머신에서 수행되기 위한 클래스파일을 압축하여 메모리에 적재시킴으로써 메모리 소요 비용을 감소시킬 수 있는 효과가 있다.

또한, 본 발명은, 압축 코드의 표현식이 바이트코드 정의에서 내장형 자바가상머신에서 사용되지 않는 부분만을 이용하기 때문에 향후 바이트코드 명령어 집합이 확장되는 상황에서도 호환성을 가지고 적용될 수 있는 효과가 있다.

(57) 청구의 범위

청구항 1.

자바 시스템에 적용되는 내장형 자바가상머신을 위한 바이트코드 압축 방법에 있어서,

클래스파일의 상기 바이트코드에서 반복되는 기본 코드 블록을 확인하여 사전에 저장하는 제 1 단계; 및

상기 기본 코드 블록 위치에 상기 사전에 저장된 상기 기본 코드 블록을 가르키는 압축 코드를 위치시켜 상기 바이트코드를 압축시키는 제 2 단계

를 포함하는 내장형 자바가상머신을 위한 바이트코드 압축 방법.

청구항 2.

제 1 항에 있어서,

상기 제 1 단계는,

상기 클래스파일에서 상기 바이트코드의 점프 명령어 타겟 주소를 제외한 기본 코드 블록을 추출하여 해쉬 테이블에 저장하는 제 3 단계;

상기 해쉬 테이블에서 위치하는 상기 기본 코드 블록의 이득을 확인하는 제 4 단계; 및

상기 해쉬 테이블에 위치하는 상기 기본 코드 블록 중 이득이 큰 기본 코드 블록을 우선으로 하여 상기 사전에 저장하는 제 5 단계

를 포함하는 내장형 자바가상머신을 위한 바이트코드 압축 방법.

청구항 3.

제 1 항 또는 제 2 항에 있어서,

상기 제 2 단계는,

상기 바이트코드 내에서 상기 사전에 저장된 상기 기본 코드 블록을 삭제하는 제 6 단계; 및

상기 바이트코드 내에 상기 기본 코드 블록의 위치에 압축된 코드임을 의미하는 명령어와 상기 기본 코드 블록의 상기 사전에서의 위치를 나타내는 인덱스 정보로 된 상기 압축 코드를 위치시켜 상기 바이트코드를 압축시키는 제 7 단계

를 포함하는 내장형 자바가상머신을 위한 바이트코드 압축 방법.

청구항 4.

제 3 항에 있어서,

상기 압축된 코드임을 의미하는 명령어는,

상기 내장형 자바가상머신에서는 사용하지 않는 퀵(quick) 명령어인 것을 특징으로 하는 내장형 자바가상머신을 위한 바이트코드 압축 방법.

청구항 5.

제 4 항에 있어서,

상기 퀵(quick) 명령어는,

전체 퀵(quick) 명령어 중 압축된 코드 블록이 끝나는 정보임을 나타내는 퀵 리턴(quick return) 명령어를 제외한 나머지 퀵(quick) 명령어인 것을 특징으로 하는 내장형 자바가상머신을 위한 바이트코드 압축 방법.

청구항 6.

제 5 항에 있어서,

상기 바이트코드를 디코딩하면서 압축코드임을 나타내는 상기 퀵(quick) 명령어를 만나면 상기 사전에서 해당 내용을 읽어 디코딩하고 상기 압축코드의 종료를 나타내는 상기 퀵 리턴(quick return) 명령어를 만나면 상기 바이트코드로 돌아와 디코딩을 수행하여 상기 바이트코드를 실행시키는 제 8 단계

를 더 포함하는 내장형 자바가상머신을 위한 바이트코드 압축 방법.

청구항 7.

대용량 프로세서를 구비한 자바 시스템에,

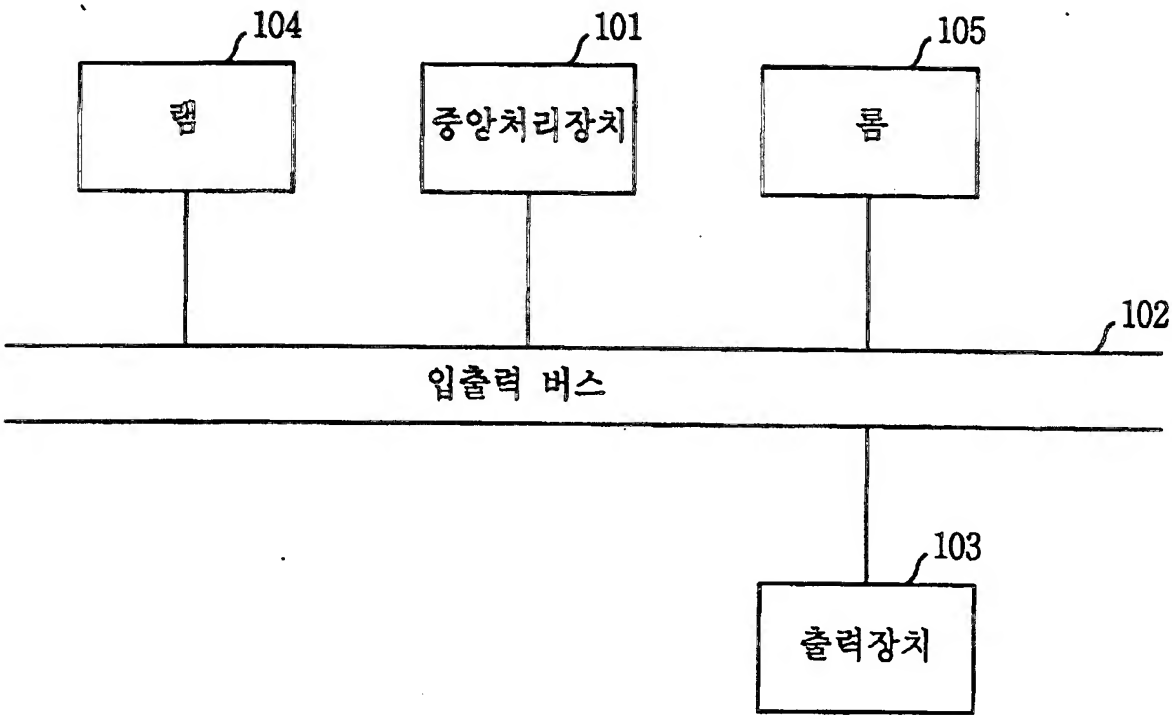
클래스파일의 상기 바이트코드에서 반복되는 기본 코드 블록을 확인하여 사전에 저장하는 제 1 기능; 및

상기 기본 코드 블록 위치에 상기 사전에 저장된 상기 기본 코드 블록을 가르키는 압축 코드를 위치시켜 상기 바이트코드를 압축시키는 제 2 기능

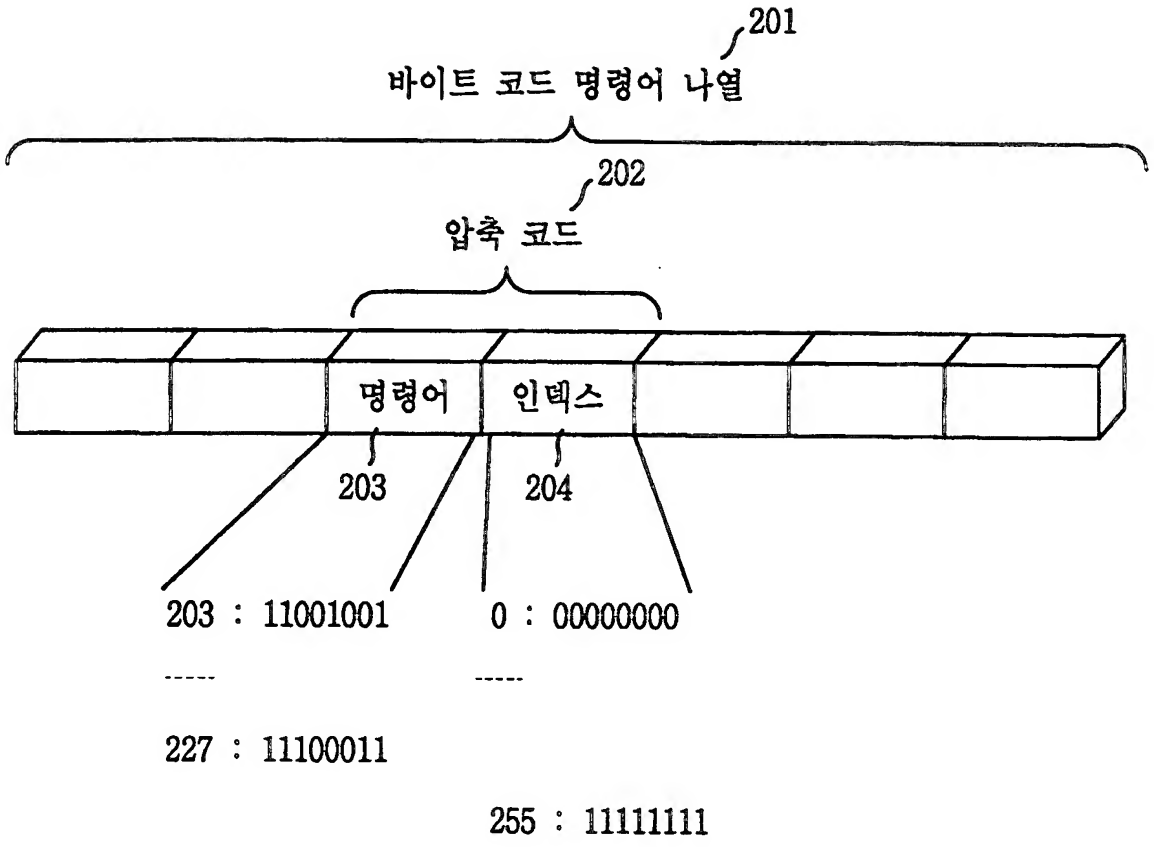
을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

도면

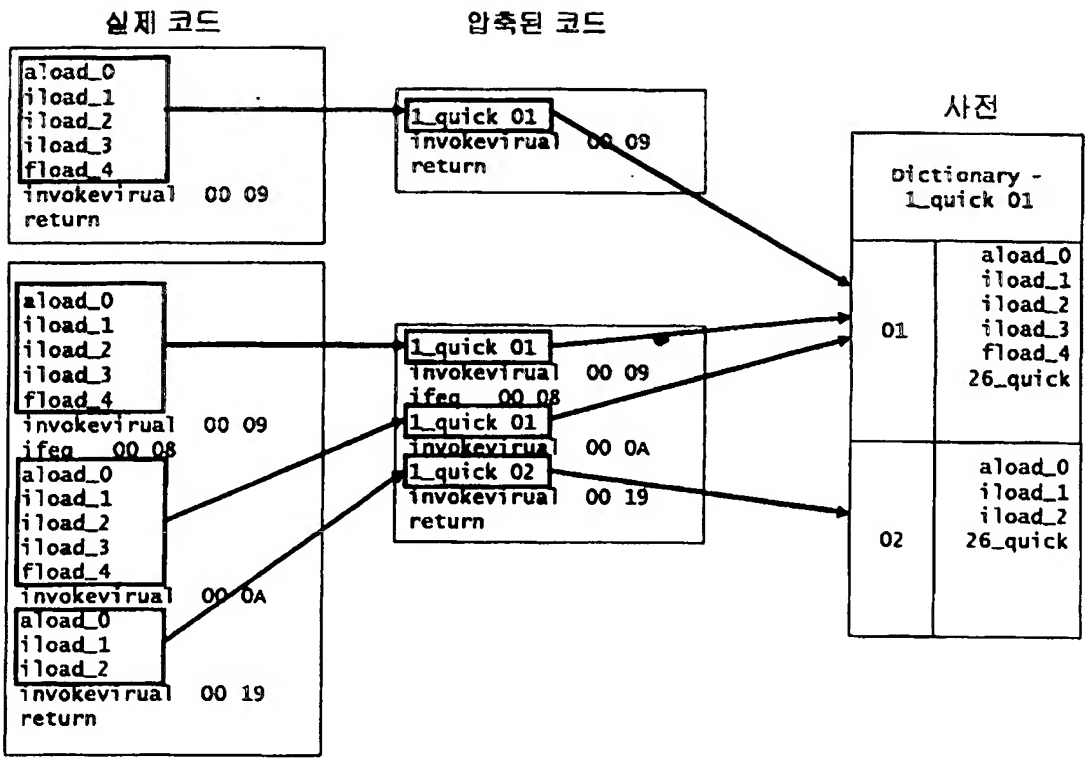
도면 1



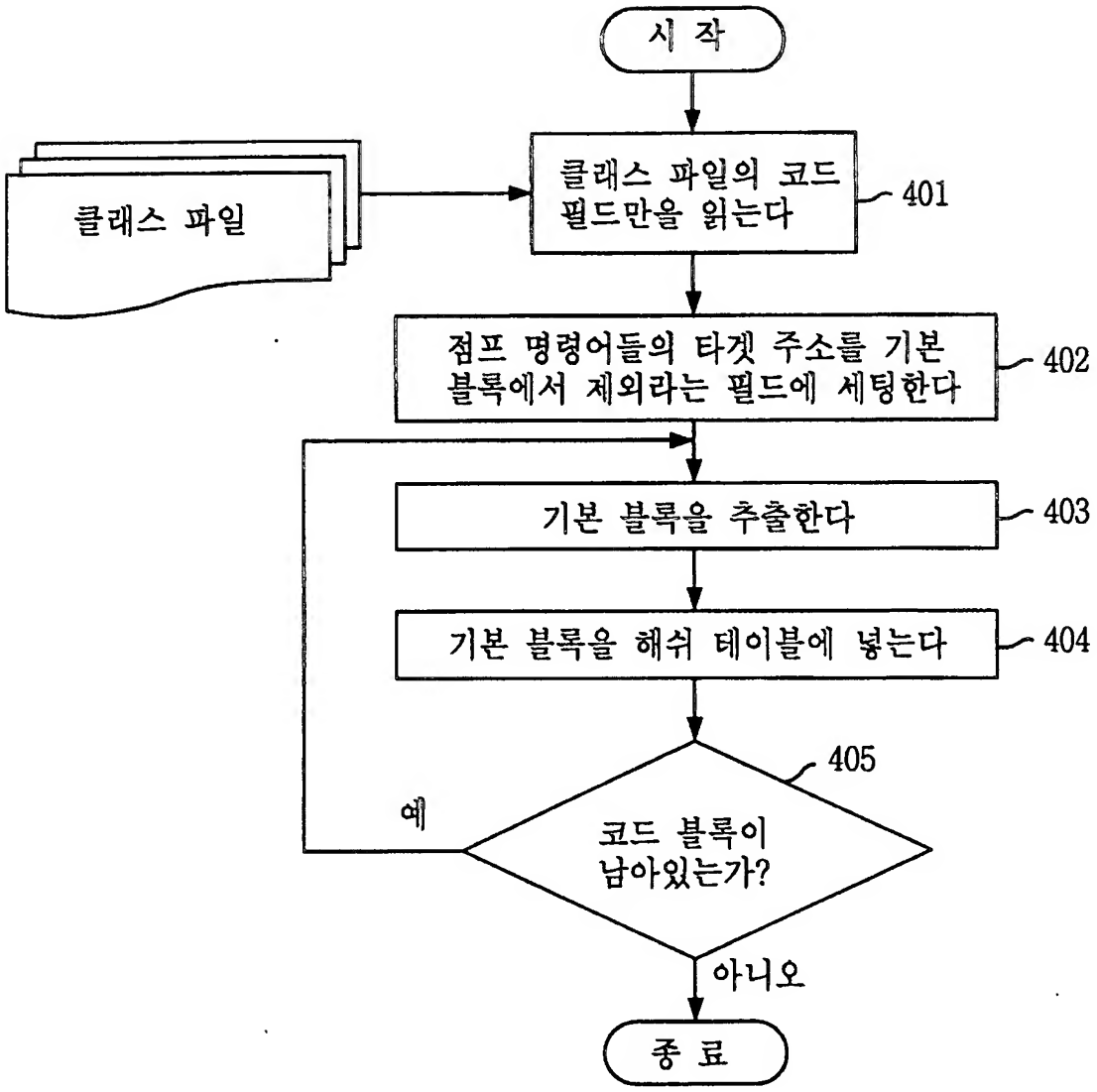
도면 2


201
바이트 코드 명령어 나열
202
압축 코드
명령어
인덱스
203
204
203 : 11001001
0 : 00000000
227 : 11100011
255 : 11111111

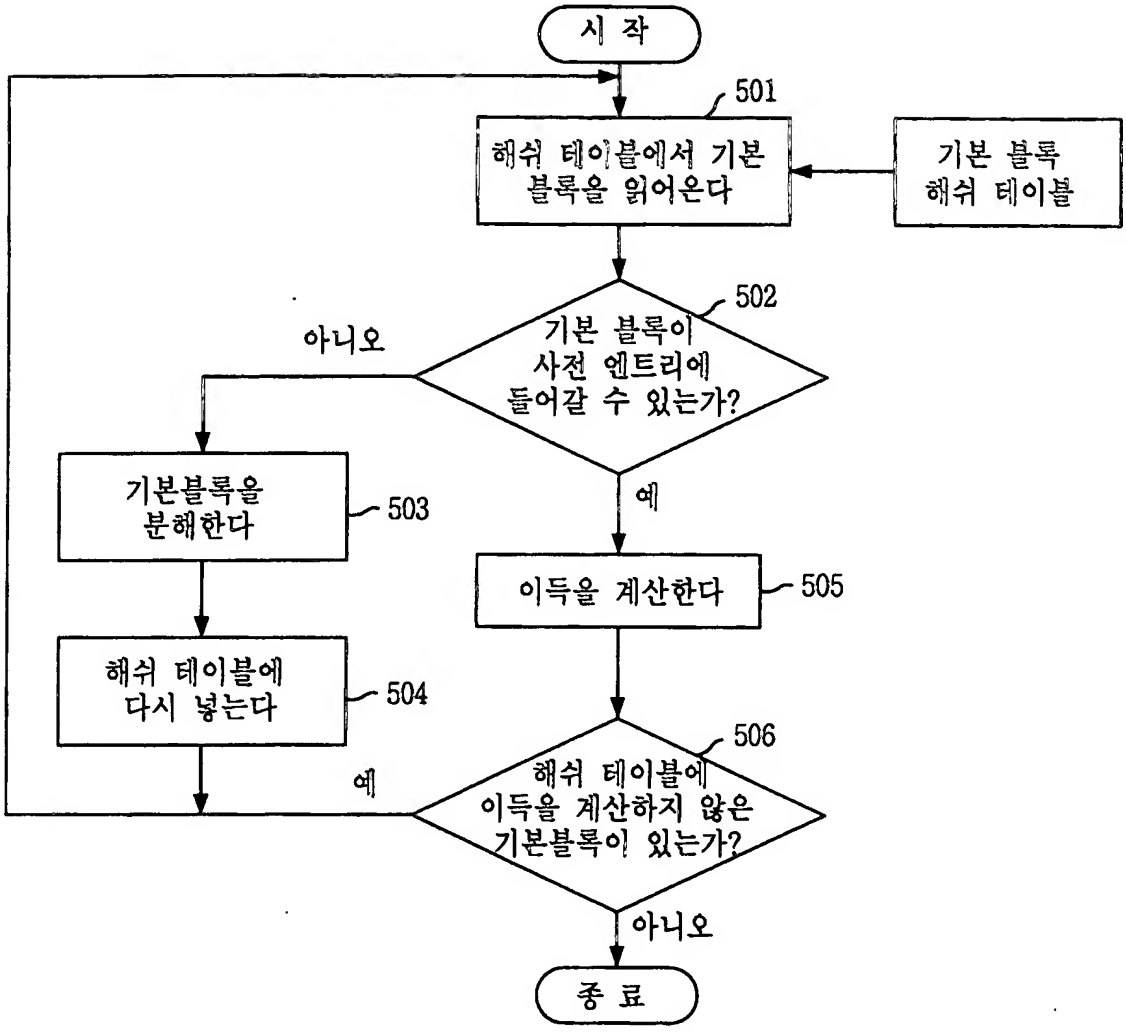
도면 3


실제 코드
압축된 코드
사전
aload_0
iload_1
iload_2
iload_3
fload_4
invokevirual 00 09
return
1_quick 01
invokevirual 00 09
return
Dictionary - 1_quick 01
01
aload_0
iload_1
iload_2
iload_3
fload_4
26_quick
ifeq 00 08
1_quick 01
invokevirual 00 0A
1_quick 02
invokevirual 00 19
02
aload_0
iload_1
iload_2
26_quick

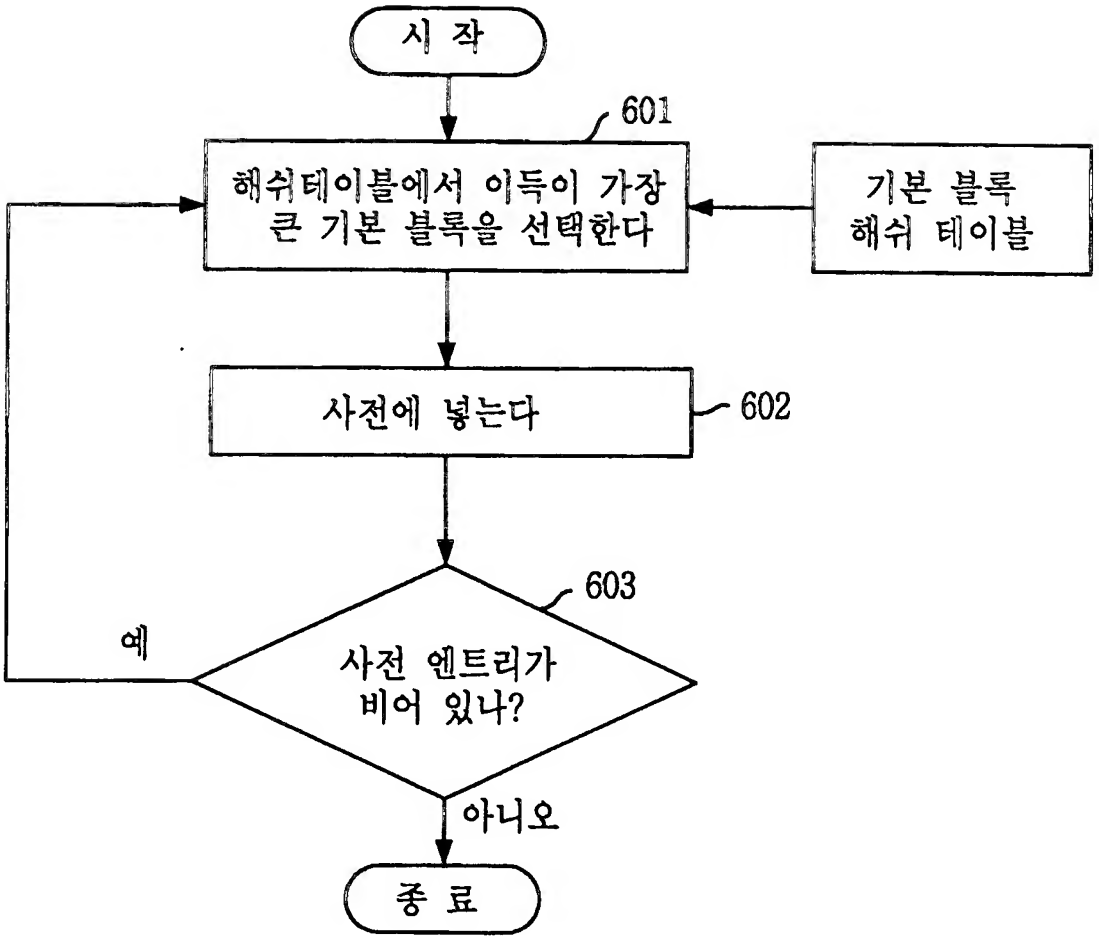
도면 4


시 작
클래스 파일
클래스 파일의 코드 필드만을 읽는다
401
점프 명령어들의 타겟 주소를 기본 블록에서 제외라는 필드에 세팅한다
402
기본 블록을 추출한다
403
기본 블록을 해쉬 테이블에 넣는다
404
405
코드 블록이 남아있는가?
예
아니오
종 료

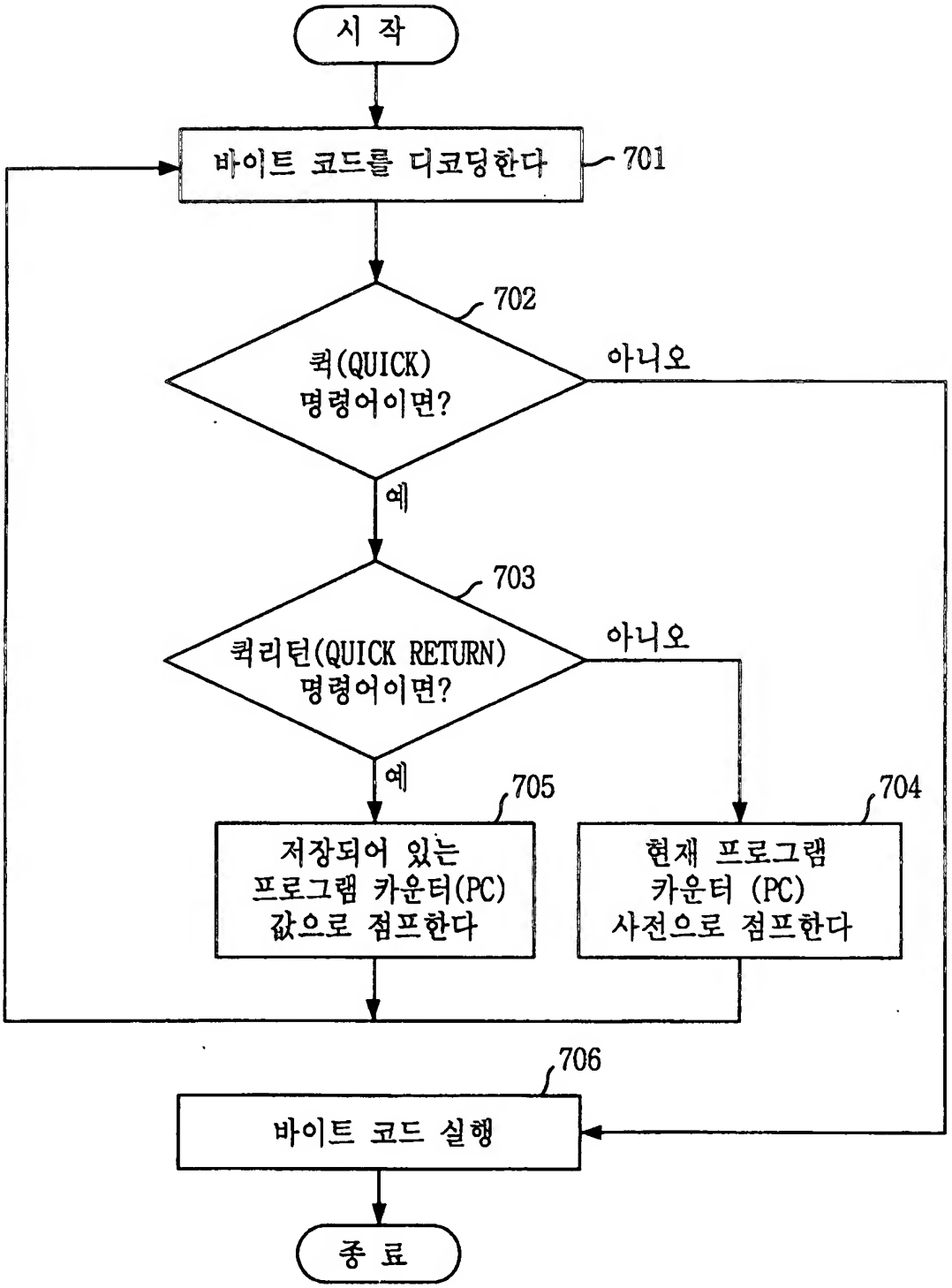
도면 5


시 작
501
해쉬 테이블에서 기본 블록을 읽어온다
기본 블록 해쉬 테이블
502
기본 블록이 사전 엔트리에 들어갈 수 있는가?
아니오
예
503
기본블록을 분해한다
504
해쉬 테이블에 다시 넣는다
505
이득을 계산한다
506
해쉬 테이블에 이득을 계산하지 않은 기본블록이 있는가?
예
아니오
종 료

도면 6


시 작
601
해쉬테이블에서 이득이 가장 큰 기본 블록을 선택한다
기본 블록 해쉬 테이블
사전에 넣는다
602
603
예
사전 엔트리가 비어 있나?
아니오
종 료

도면 7


시 작
바이트 코드를 디코딩한다
701
702
퀵(QUICK)
명령어이면?
아니오
예
703
퀵리턴(QUICK RETURN)
명령어이면?
아니오
예
705
저장되어 있는
프로그램 카운터(PC)
값으로 점프한다
704
현재 프로그램
카운터 (PC)
사전으로 점프한다
706
바이트 코드 실행
종 료